模拟学生和老师的一天

学生	老师
人物出场介绍（姓名、年龄、年级）	人物出场介绍（姓名、年龄、部门）
起床（睁开眼睛、起身、穿好衣服）	起床（睁开眼睛、起身、穿好衣服）
洗漱（刷牙、洗脸）	洗漱（刷牙、洗脸）
吃饭（吃菜、扒饭）	吃饭（吃菜、扒饭）
登录账号（输入账号密码、登录成功）	打卡（录入指纹、打卡成功）
学习（看视频、查资料、写代码）	工作（授课、答疑、写代码）
吃饭（吃菜、扒饭）	吃饭（吃菜、扒饭）
学习（看视频、查资料、写代码）	工作（授课、答疑、写代码）
吃饭（吃菜、扒饭）	吃饭（吃菜、扒饭）
洗漱（刷牙、洗脸）	洗漱（刷牙、洗脸）
睡觉（脱掉外套、躺下、闭眼）	睡觉（脱掉外套、躺下、闭眼）
人数统计（统计、公布）	人数统计（统计、公布）

面向过程编程

```python
def get_up(name):
    print(f'{name}睁开眼睛')
    print(f'{name}起身')
    print(f'{name}穿好衣服')


def wash(name):
    print(f'{name}刷牙')
    print(f'{name}洗脸')


def eat(name):
    print(f'{name}吃菜')
    print(f'{name}扒饭')


def login_id(name):
    print(f'{name}输入账号密码')
    print(f'{name}登陆账号成功')


def study(name):
    print(f'{name}看视频')
    print(f'{name}查资料')
    print(f'{name}写代码')


def sleep(name):
    print(f'{name}脱掉外套')
```

```
    print(f'{name}躺下')
    print(f'{name}闭上眼睛')


count_s = 0
stu1 = '张三'
age1 = 18
grade1 = '高三'
print(f'大家好! 我是{stu1}, 今年{age1}岁, 目前正在读
{grade1}!')
count_s += 1
get_up(stu1)
wash(stu1)
eat(stu1)
login_id(stu1)
study(stu1)
eat(stu1)
study(stu1)
eat(stu1)
wash(stu1)
sleep(stu1)
print(f'当前统计的学生人数为: {count_s}')
```

```
def get_up(name):
    print(f'{name}睁开眼睛')
    print(f'{name}起身')
    print(f'{name}穿好衣服')


def wash(name):
```

```
    print(f'{name}刷牙')
    print(f'{name}洗脸')


def eat(name):
    print(f'{name}吃菜')
    print(f'{name}扒饭')


def clock_in(name):
    print(f'{name}录入指纹')
    print(f'{name}打卡成功')


def work(name):
    print(f'{name}授课')
    print(f'{name}答疑')
    print(f'{name}写代码')


def sleep(name):
    print(f'{name}脱掉外套')
    print(f'{name}躺下')
    print(f'{name}闭上眼睛')


count_t = 0
t1 = '老赵'
age1 = 39
department = '教学部'
```

```
print(f'大家好！ 我是{t1}, 今年{age1}岁, 在
{department}任职!')
count_t += 1
get_up(t1)
wash(t1)
eat(t1)
clock_in(t1)
work(t1)
eat(t1)
work(t1)
eat(t1)
wash(t1)
sleep(t1)
print(f'当前统计的老师人数为: {count_t}')
```

面向对象编程

```
class Person:

    def __init__(self, name, age):
        self.name = name
        self.age = age
        self.show_time()

    def get_up(self):
        print(f'{self.name}睁开眼睛')
        print(f'{self.name}起身')
        print(f'{self.name}穿好衣服')

    def wash(self):
```

```
        print(f'{self.name}刷牙')
        print(f'{self.name}洗脸')

    def eat(self):
        print(f'{self.name}吃菜')
        print(f'{self.name}扒饭')

    def sleep(self):
        print(f'{self.name}脱掉外套')
        print(f'{self.name}躺下')
        print(f'{self.name}闭上眼睛')

    def show_time(self):
        pass


class Student(Person):

    count_s = 0

    def __init__(self, name, age, grade):
        self.grade = grade
        super(Student, self).__init__(name, age)
        Student.count_s += 1

    def show_time(self):
        print(f'大家好! 我是{self.name}, 今年
{self.age}岁, 目前正在读{self.grade}!')

    def login_id(self):
```

```
        print(f'{self.name}输入账号密码')
        print(f'{self.name}登陆账号成功')

    def study(self):
        print(f'{self.name}看视频')
        print(f'{self.name}查资料')
        print(f'{self.name}写代码')

    @classmethod
    def publish(cls):
        print(f'当前统计的学生人数为:
{cls.count_s}')


class Teacher(Person):

    count_t = 0

    def __init__(self, name, age, department):
        self.department = department
        super(Teacher, self).__init__(name, age)
        Teacher.count_t += 1

    def show_time(self):
        print(f'大家好! 我是{self.name}, 今年
{self.age}岁, 在{self.department}任职!')

    def clock_in(self):
        print(f'{self.name}录入指纹')
        print(f'{self.name}打卡成功')
```

```
    def work(self):
        print(f'{self.name}授课')
        print(f'{self.name}答疑')
        print(f'{self.name}写代码')

    @classmethod
    def publish(cls):
        print(f'当前统计的老师人数为:
{cls.count_t}')


stu1 = Student('张三', 18, '高三')
stu2 = Student('李四', 16, '高一')
stu3 = Student('王五', 17, '高二')
Student.publish()

t1 = Teacher('老赵', 39, '教学部')
t2 = Teacher('老孙', 45, '后勤部')
Teacher.publish()
```

面向对象基本概念

类对象、实例对象、类属性、实例属性

```
"""
类，类对象
object是所有类的父类，通常省略不写
"""
class Student(object):

    school = '深兰教育'  # 类属性(类变量)

    def __init__(self, name, age):
        self.name = name  # 实例属性(实例变量)
        self.age = age



"""
魔术方法(特殊方法)：官方定义好的，以两个下划线开头并且以两个下划线结尾来命名的方法
魔术方法特点：一般不需要主动调用，在满足特定条件时，会被自动调用

__new__称为构造方法，用来创建实例对象，并返回该实例对象
__init__称为初始化方法，可以对实例对象进行属性定制，没有返回值

每当实例化时，先自动调用魔术方法__new__(cls, *args, **kwargs)，把要实例化的类对象(即：Student)作为实参传递给形参cls，并把实例化时传入的其他实参(即：'张三', 28)传递给形参*args, **kwargs，然后__new__方法根据cls创建出一个对应的实例对象，并返回该实例对象(即：stu1=该实例对象)
```

```
再自动调用魔术方法__init__(self, name, age), 把__new__方法创建的实例对象(即: stu1)作为实参传递给形参self, 实例化时传入的其他实参(即: '张三', 28)分别传给形参name, age, 然后__init__方法再对self进行属性定制(inplace操作)
"""
stu1 = Student('张三', 28)
stu2 = Student('李四', age=32)

""" 调用实例属性: 只能用实例对象调用, 不能用类对象调用
"""
print(stu1.name)
print(stu2.name)

print(getattr(stu1, 'age'))
print(getattr(stu1, 'adres', '该实例属性不存在'))

"""
调用类属性: 既可以用类对象调用(推荐), 也可以用实例对象调用
注意: 当实例属性和类属性同名时, 实例对象优先调用实例属性
"""
print(Student.school)
print(stu1.school)
print(stu2.school)

print(getattr(Student, 'school'))
print(getattr(Student, 'adres', '该类属性不存在'))
```

```
""" 修改实例属性: 只能用实例对象修改 """
stu1.age = 29
print(stu1.age)

setattr(stu1, 'age', 27)
print(stu1.age)

""" 修改类属性: 只能用类对象修改 """
Student.school = '深兰大学'
print(Student.school)

setattr(Student, 'school', '深兰教育')
print(Student.school)

"""
动态定义实例属性: 当实例对象修改的属性不存在时, 则新增该实例属性
"""
stu1.school = 'ShenLanEdu'
print(stu1.school)  # 给stu1新增一个实例属性
print(Student.school)  # 类属性不变

setattr(stu2, 'adres', '威宁路')
print(stu2.adres)  # 给stu2新增一个实例属性

""" 动态定义类属性: 当类对象修改的属性不存在时, 则新增该类属性 """
Student.subject = 'AI'
print(Student.subject)  # 新增一个类属性
print(stu1.subject)
```

```
print(stu2.subject)

setattr(Student, 'course', '人工智能')
print(Student.course)  # 新增一个类属性
print(stu1.course)
print(stu2.course)

""" 删除属性: 可以用del语句 """
del stu1.age
delattr(stu1, 'name')

del Student.school
delattr(Student, 'subject')

""" 判定属性是否存在 """
print(hasattr(Student, 'school'))
print(hasattr(stu1, 'name'))
print(hasattr(stu2, 'age'))
```

与属性操作相关的内置函数

delattr(object, name)

- 删除 object 的 name 属性（name 参数为字符串）

```
class Person:

    eat = "rice"
```

```
    def __init__(self, age):
        self.age = age


p = Person(18)
print(Person.eat)
""" 等价于 del Person.eat """
delattr(Person, "eat")  # 删除类属性eat
print(Person.eat)


print(p.age)
""" 等价于 del p.eat """
delattr(p, "age")  # 删除实例属性age
print(p.age)
```

getattr(object, name[, default])

- 返回 object 对象的 name 属性值（name 参数为字符串）
- 如果 name 属性不存在，且提供了 default 值，则返回它，否则触发 AttributeError

```
class Person:

    eat = "rice"

    def __init__(self, age):
        self.age = age
```

```
p = Person(18)

""" 等价于 Person.eat """
print(getattr(Person, "eat"))

""" 等价于 p.age """
print(getattr(p, "age"))

print(getattr(p, "height", 178))
# print(getattr(p, "height"))
```

hasattr(object, name)

- 判断 object 对象是否包含 name 属性（name 参数为字符串），返回 True 或 False
- 此功能是通过调用 getattr(object, name) 看是否有 AttributeError 异常来实现的

```
class Person:

    eat = "rice"

    def __init__(self, age):
        self.age = age


p = Person(18)
print(hasattr(Person, "eat"))  # True
print(hasattr(p, "eat"))  # True
print(hasattr(p, "age"))  # True
print(hasattr(p, "height"))  # False
```

setattr(object, name, value)

- 将 object 的 name 属性设置为 value，属性不存在则新增属性（name参数为字符串）

```
class Person:

    eat = "rice"

    def __init__(self, age):
        self.age = age


p = Person(18)
setattr(Person, "eat", "noodles")
print(Person.eat)
```

```
setattr(Person, "drink", "water")
print(Person.drink)

setattr(p, "age", 29)
print(p.age)

setattr(p, "height", 178)
print(p.height)
```

类方法、对象方法、静态方法

- 通常把定义在类中的函数叫方法（method）
- 对象方法的第一个参数位隐式的接收了实例对象
- 类方法的第一个参数位隐式的接收了类对象

```
class Student:

    school = '深兰教育'

    def __init__(self, name):
        self.name = name

    def study1(self, course):  # 对象方法
        print(f'{self.name}在学习{course}课!')

    @classmethod  # 类方法装饰器
```

```
    def study2(cls, course):
        print(f'{cls.school}的学生在学习{course}
课!')
        print(f'{Student.school}的学生在学习
{course}课!')

    @staticmethod  # 静态方法装饰器
    def study3(course):
        print(f'{Student.school}的学生在学习
{course}课!')

    @property  # 只读属性装饰器
    def study4(self):
        return f'{self.name}在学习Python课'


stu1 = Student('张三')
stu2 = Student('李四')

"""
调用对象方法: 通常用实例对象去调用, 用类对象调用时需要主
动给self传实参
"""
stu1.study1('Python')
stu2.study1('机器学习')
Student.study1(stu1, 'Python')
Student.study1(stu2, '机器学习')

"""
```

```
调用类方法: 既可以用类对象调用(推荐), 也可以用实例对象调用
"""
Student.study2('Python')
stu1.study2('Python')
stu2.study2('Python')

"""
调用静态方法: 既可以用类对象调用(推荐), 也可以用实例对象调用
"""
Student.study3('Python')
stu1.study3('Python')
stu2.study3('Python')

"""
调用只读属性: 用实例对象调用
"""
print(stu1.study4)
print(stu2.study4)
```

面向对象三大特性

封装

- 在属性名或方法名前面加两个下划线开头, 声明为私有属性或私有方法
- 私有属性或私有方法只能在该类的内部调用, 不能在该类的外部直接调用

```
class Person:

    school = '深兰教育'
    __eat = 'rice'

    def __init__(self, name, age):
        self.name = name
        self.__age = age

    def get_up(self):
        print(f'{self.name}起床了!')

    def __sleep(self):
        print(f'{self.name}睡觉了!')

    @classmethod
    def get_eat(cls):
        return cls.__eat

    def get_age(self):
        return self.__age

    def call_sleep(self):
        self.__sleep()
```

```
print(Person.school)
# Person.__eat  # Error
print(Person.get_eat())


p1 = Person('张三', 19)
print(p1.name)
# p1.__age  # Error
print(p1.get_age())


p1.get_up()
# p1.__sleep()  # Error
p1.call_sleep()
```

继承

- 所有的类都默认继承内置的 object 类，通常不用显式的写出来
- 子类继承父类后，就可以调用父类中的属性和方法

```python
class A:  # 父类
    pass


class B(A):  # 子类
    pass
```

单继承

- 继承顺序：先找当前类，再找父类，再找父类的父类，依此类推

```python
class Person:

    state = "China"

    @staticmethod
    def eat():
        print('吃饭')

    @staticmethod
    def speak():
        print('说话')


class Student(Person):

    @staticmethod
```

```
    def study():
        print('读书')


class Worker(Person):

    @staticmethod
    def work():
        print('搬砖')


Student.study()
Student.eat()
Student.speak()
print(Student.state)

Worker.work()
Worker.eat()
Worker.speak()
print(Worker.state)
```

```
class Animal:

    @staticmethod
    def eat():
        print('吃东西')


class Cat(Animal):
```

```
    @staticmethod
    def catch_mouse():
        print('抓老鼠')


class Ragdoll(Cat):

    @staticmethod
    def cute():
        print('卖萌')


Ragdoll.cute()
Ragdoll.catch_mouse()
Ragdoll.eat()
```

多重继承

- 继承顺序：先找当前类，再按照从左往右的顺序依次找对应的父类

```
class Animal:

    @staticmethod
    def eat():
        print('吃东西')
```

```
class Cat:

    @staticmethod
    def catch_mouse():
        print('抓老鼠')


class Ragdoll(Cat, Animal):

    @staticmethod
    def cute():
        print('卖萌')


Ragdoll.cute()
Ragdoll.catch_mouse()
Ragdoll.eat()
```

方法重写

- 在继承关系中，当父类的方法不能满足子类的需求时，可以在子类重写父类的该方法

```
class Animal:

    def __init__(self, food):
        self.food = food
```

```
    def eat(self):
        print(f"动物吃{self.food}")


class Cat(Animal):

    # 为了实现'猫吃鱼'的功能，而不是父类的'动物吃鱼'
    # 子类对eat方法重写
    def eat(self):
        print(f"猫吃{self.food}")


c = Cat("鱼")
c.eat()
```

super()

- super是内置的类，可以表示指定类的父类（超类）
- 适用场景：在子类重写父类方法后，想再调用父类的该方法

```
class Animal:

    def eat(self):
        print("吃东西")


class Cat(Animal):

```

```
    def eat(self):
        print("吃鱼")


class Ragdoll(Cat):

    def eat(self):
        print("喝咖啡")


rd = Ragdoll()
rd.eat()
super(Ragdoll, rd).eat()  # rd调用Ragdoll父类的eat方法
super(Cat, rd).eat()  # rd调用Cat父类的eat方法

c = Cat()
c.eat()  # c调用Cat类中的eat方法
super(Cat, c).eat()  # c调用Cat父类的eat方法
```

继承中的**__init__**方法

```
class A:

    def __init__(self, name):
        self.name = name
        self.Q()
```

```
    def Q(self):
        print(self.name, 'Q方法被调用')


class B(A):
    pass


b = B('张三')
b.Q()


class C(A):

    def __init__(self, name):
        self.name = name


c = C('赵六')
c.Q()


class D(A):

    def __init__(self, name):
        super(D, self).__init__('李四')
        self.name = name


d = D('王五')
```

```
d.Q()
```

与继承相关的两个内置函数

isinstance(object, classinfo)

- object：实例对象
- classinfo：类对象或者由多个类对象构成的元组
- 判定 object 是否为 classinfo 的实例对象或者其子类的实例对象

```
class A:
    pass


class B(A):
    pass


class C(A):
    pass


a = A()
b = B()
c = C()
print(isinstance(a, A))  # True
print(type(a) == A)  # True
```

```
print(isinstance(b, A))  # True，考虑继承
print(type(b) == A)  # False，type不考虑继承
print(isinstance(c, A))  # True，考虑继承
print(type(c) == A)  # False，type不考虑继承
print(isinstance(c, (B, A)))  # True，c是A子类的实例
```

issubclass(class, classinfo)

- class：类对象
- classinfo：类对象或者由多个类对象构成的元组
- 判定 class 是否为 classinfo 的子类
- 该函数会把自己视作为自己的子类

```
class A:
    pass


class B(A):
    pass


class C(A):
    pass


print(issubclass(B, A))  # True
print(issubclass(C, A))  # True
print(issubclass(A, A))  # True，类会被视作其自身的子类
```

```
print(issubclass(C, (B, A)))  # True
```

多态性

- 多态性是指具有不同内容的方法可以使用相同的方法名，则可以用一个方法名调用不同内容的方法

```
class Apple:

    @staticmethod
    def change():
        return '啊~ 我变成了苹果汁!'


class Banana:

    @staticmethod
    def change():
        return '啊~ 我变成了香蕉汁!'


class Mango:

    @staticmethod
    def change():
        return '啊~ 我变成了芒果汁!'
```

```
class Juicer:

    @staticmethod
    def work(fruit):
        print(fruit.change())


"""
三个内容不同的change方法使用相同的名字命名，
只要改变change的调用对象，就可以调用不同内容的方法
"""
Juicer.work(Apple)
Juicer.work(Banana)
Juicer.work(Mango)
```